Stepper Input

Stepper A sophisticated Angular multi-step navigation component for creating step-by-step wizards, form flows, and progress workflows. Features animated progress indicators, interactive navigation, and flexible layout options for guiding users through complex processes. How to use ■ import { AvaStepperComponent } from "@aava/play-core" ; Basic Usage ■ The most basic implementation with step labels and default horizontal orientation. Sizes ■ Three size options to accommodate different interface densities and visual hierarchies. Available Sizes ■ xs : Extra small for minimal interfaces sm : Small for compact layouts md : Medium size for most use cases (default) lg : Large for prominent workflows and accessibility Icon Variants ■ Enhanced stepper with custom icons for each step, providing better visual context and user guidance. Icon Features ■ Custom Step Icons : Replace numeric indicators with meaningful Lucide icons Contextual Guidance : Icons provide immediate visual context for each step State Awareness : Icons adapt to active, completed, and disabled states Consistent Sizing : Icons scale appropriately with stepper size variants Color Theming : Icons inherit stepper theme colors automatically Accessibility : Proper ARIA labels and descriptions for screen readers Orientation ■ Flexible layout options for different design requirements and content arrangements. Horizontal Orientation ■ Default layout : Steps arranged left to right Space efficient : Ideal for wide containers Progress visualization : Horizontal progress lines Label positioning : Labels positioned below step circles Vertical Orientation ■ Alternative layout : Steps arranged top to bottom Sidebar friendly : Perfect for narrow containers and sidebars Vertical progress : Connecting lines flow downward Inline labels : Labels positioned next to step circles Interactive Navigation ■ Control user interaction and step accessibility with interactive navigation options. Interactive Features ■ Click navigation : Jump to any accessible step by clicking Keyboard navigation : Enter and Space key support Disabled steps : Prevent navigation to specific steps Step validation : Control progression based on form validity Non-interactive mode : Display-only stepper for progress indication Event Handling ■ Comprehensive event system for tracking step changes and workflow completion. Available Events ■ stepChange : Emitted when user navigates to a different step stepperComplete : Emitted when the final step is reached Step validation : Handle step transitions with custom validation logic Accessibility ■ Built-in accessibility features ensuring WCAG compliance and inclusive user experience. Accessibility Features ■ Keyboard navigation : Tab, Enter, and Space key support ARIA labels : Descriptive labels for screen readers Role attributes : Proper button roles for interactive elements Focus management : Clear visual focus indicators Step announcements : Screen reader notifications for step changes High contrast : Support for high contrast mode preferences API Reference ■ Inputs ■ Property Type Default Description steps (string | StepperStep)[] [] Array of step labels or step objects to display currentStep number 0 Index of the currently active step (0-based) orientation 'horizontal' | 'vertical' 'horizontal' Layout orientation of the stepper showNavigation boolean true Whether to show navigation elements interactive boolean true Whether steps are clickable for navigation size 'xs' | 'sm' | 'md' | 'lg' 'md'

Visual size of the stepper component disabledSteps number[] [] Array of step indices that should be disabled iconColor string '#fff' Color for the check mark icons in completed steps iconSize string '20' Size of the check mark icons stepVariant 'default' | 'icon' 'default' Variant of step display (default numbers or icons) showLabel boolean true Whether to show step labels below circles customStyles Record<string, string> {} Custom CSS styles to apply to the stepper Outputs ■ Event Type Description stepChange EventEmitter<number> Emitted when user navigates to a different step stepperComplete EventEmitter<void> Emitted when the workflow reaches completion Methods ■ Method Parameters Return Type Description goToStep(index: number) index: number void Navigate to a specific step programmatically isDisabled(index: number) index: number boolean Check if a specific step is disabled CSS Custom Properties ■ Property Default Description --stepper-wrapper-background Dynamic Background color for active/completed elements --stepper-background Dynamic Background color for inactive elements --stepper-line-completed-background Dynamic Background for completed progress lines --step-label-font Dynamic Font size for step labels --step-label-active-font-weight Dynamic Font weight for active/completed labels --step-circle-text Dynamic Text color for step circles --stepper-size-sm-circle-size Dynamic Circle size for small variant --stepper-size-md-circle-size Dynamic Circle size for medium variant --stepper-size-lg-circle-size Dynamic Circle size for large variant --stepper-size-sm-font Dynamic Font size for small variant --stepper-size-md-font Dynamic Font size for medium variant --stepper-size-lg-font Dynamic Font size for large variant Accessibility Guidelines ■ Keyboard Navigation ■ Tab : Navigate to stepper and move between interactive elements Enter : Activate focused step (if interactive) Space : Activate focused step (if interactive) Arrow Keys : Alternative navigation between steps Screen Reader Support ■ Use descriptive step labels that clearly indicate the purpose Provide context about the total number of steps Announce step changes and progress updates Include completion status in step descriptions Use appropriate heading levels for step content Visual Design ■ Maintain sufficient color contrast (4.5:1 minimum) for all states Provide clear focus indicators on interactive elements Ensure step circles meet minimum touch target size (44px) Use consistent visual hierarchy across step states Support high contrast and reduced motion preferences Best Practices ■ Design Guidelines ■ Use clear, action-oriented step labels Keep step labels concise but descriptive Provide visual feedback for all step states Consider orientation based on layout constraints Group related steps logically in the workflow Performance ■ Minimize step array modifications to prevent re-rendering Use OnPush change detection strategy for optimal performance Debounce rapid step changes if triggered programmatically Optimize animations for lower-end devices

```
<aava-stepper
  [steps]="steps"
  [currentStep]="currentStep"
  orientation="horizontal"
  size="md"
  [iconColor]="'#fff'"
  [iconSize]="'20'"
  (stepChange)="onStepChange($event)"
>
</aava-stepper>


---

steps = ['Step 1', 'Step 2', 'Step 3', 'Step 4', 'Step 5'];
currentStep = 0;

onStepChange(step: number): void {
  this.currentStep = step;
}
```

```
<aava-stepper
  [steps]="xsmallSteps"
  [currentStep]="xsmallCurrentStep"
  orientation="horizontal"
  size="xs"
  [iconColor]="'#fff'"
  [iconSize]="'16'"
  (stepChange)="onXsmallStepChange($event)"
>
</aava-stepper>

<aava-stepper
  [steps]="smallSteps"
  [currentStep]="smallCurrentStep"
  orientation="horizontal"
  size="sm"
  [iconColor]="'#fff'"
  [iconSize]="'16'"
  (stepChange)="onSmallStepChange($event)"
>
</aava-stepper>

<aava-stepper
  [steps]="mediumSteps"
  [currentStep]="mediumCurrentStep"
  orientation="horizontal"
  size="md"
  [iconColor]="'#fff'"
  [iconSize]="'20'"
  (stepChange)="onMediumStepChange($event)"
>
</aava-stepper>

<aava-stepper
  [steps]="largeSteps"
  [currentStep]="largeCurrentStep"
  orientation="horizontal"
  size="lg"
  [iconColor]="'#fff'"
  [iconSize]="'24'"
  (stepChange)="onLargeStepChange($event)"
>
</aava-stepper>


---

xsmallSteps = ['One', 'Two', 'Three', 'Four', 'Five'];
xsmallCurrentStep = 0;

smallSteps = ['One', 'Two', 'Three', 'Four', 'Five'];
smallCurrentStep = 0;

mediumSteps = ['Goal', 'Steps', 'Progress', 'Review', 'Finish'];
mediumCurrentStep = 0;
```

```
largeSteps = ['Research', 'Define', 'Ideate', 'Prototype', 'Test'];
largeCurrentStep = 0;

onXsmallStepChange(step: number): void {
  this.xsmallCurrentStep = step;
}

onSmallStepChange(step: number): void {
  this.smallCurrentStep = step;
}

onMediumStepChange(step: number): void {
  this.mediumCurrentStep = step;
}

onLargeStepChange(step: number): void {
  this.largeCurrentStep = step;
}



<aava-stepper
  [steps]="checkoutSteps"
  [currentStep]="checkoutStep"
  [stepVariant]="'icon'"
  orientation="horizontal"
  size="md"
  [iconColor]="'#fff'"
  [iconSize]="'16'"
  (stepChange)="onCheckoutStepChange($event)"
></aava-stepper>


---

checkoutSteps: StepperStep[] = [
  { label: 'Login', iconName: 'user' },
  { label: 'Shipping', iconName: 'truck' },
  { label: 'Payment', iconName: 'credit-card' },
  { label: 'Confirmation', iconName: 'check-circle' },
];
checkoutStep = 0;

onCheckoutStepChange(step: number): void {
  this.checkoutStep = step;
}
```

```
<aava-stepper
  [steps]="horizontalSteps"
  [currentStep]="horizontalCurrentStep"
  orientation="horizontal"
  size="md"
  [iconColor]="'#fff'"
  [iconSize]="'20'"
  (stepChange)="onHorizontalStepChange($event)"
>
</aava-stepper>

<aava-stepper
  [steps]="verticalSteps"
  [currentStep]="verticalCurrentStep"
  orientation="vertical"
  size="md"
  [iconColor]="'#fff'"
  [iconSize]="'20'"
  (stepChange)="onVerticalStepChange($event)"
>
</aava-stepper>


---


 horizontalSteps = [
    'Personal Info',
    'Contact Details',
    'Preferences',
    'Review',
    'Submit',
  ];
  horizontalCurrentStep = 0;

  verticalSteps = [
    'Account Setup',
    'Profile Info',
    'Preferences',
    'Confirmation',
  ];
  verticalCurrentStep = 0;

  onHorizontalStepChange(step: number) {
    this.horizontalCurrentStep = step;
    console.log('Horizontal step changed to:', step + 1);
  }

  onVerticalStepChange(step: number) {
    this.verticalCurrentStep = step;
    console.log('Vertical step changed to:', step + 1);
  }
```

■ No code found

■ No code found

■ No code found